Práctica 3
Implementación de un servicio telemático basado en webcam

1.

Lo primero que debemos de hacer para comenzar con la práctica es realizar la instalación del jdk de java, instalar la API de Webcam Capture e instalar un IDE.

Uso ubuntu 20.04 así que por defecto ya incluye Open JDK 11, que es una variante de código abierto de JRE y JDK.

Si no fuera el caso se puede instalar de la siguiente forma:

- **sudo apt update.**
- **sudo apt install default-jre.**

Verificamos la instalación asi:

- **java -version.**

Es posible también que necesitemos el kit de desarrollo de Java (JDK) además de JRE para compilar y ejecutar algunos programas específicos basados en java. Para instalar JDK, ejecute el siguiente comando:

- **sudo apt install default-jdk.**

2.

Como IDE voy a usar netbeans-8.2 el cual ya tengo instalado. En caso de no tenerlo, para su instalación en Ubuntu 20.04 utilizaríamos los siguientes comandos:

- **wget -c http://download.netbeans.org/netbeans/8.2/final/bundles/netbeans-8.2-linux.sh**

Hacemos el script ejecutable:

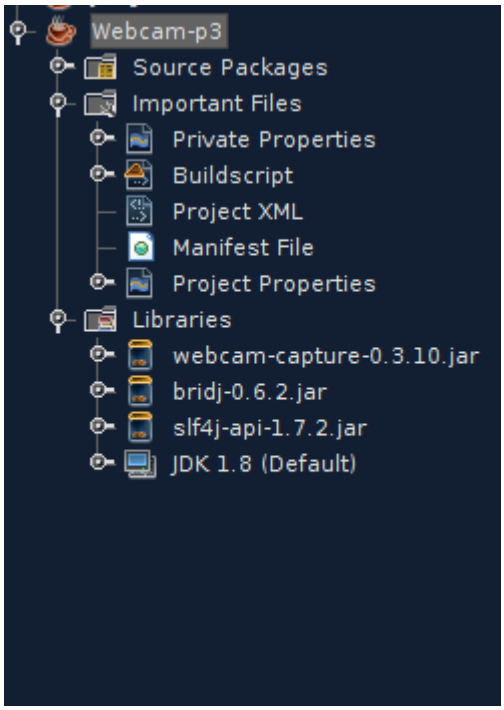
- **chmod +x netbeans-8.2-linux.sh**

- **./netbeans-8.2-linux.sh**

3.

Por último debemos instalar e incorporar los .jar de la librería de manejo de webcams del siguiente enlace → **http://webcam-capture.sarxos.pl/**

Las incorporamos a nuestro proyecto.



Servicio telemático mediante sockets TCP

ServidorTCP

```java
public class ServidorTCP {

    public static void main(String args[]) throws IOException
    {
            Webcam webcam = Webcam.getDefault();
            webcam.open();

            try(ServerSocket ss = new ServerSocket(8890)){
                while(true){
                    System.out.println("Esperando...");
                    Socket s = ss.accept();
                    BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));

                    //obtenemos la imagen y la muestra en el momento que el cliente manda una petición.
                    BufferedImage image = webcam.getImage();
                    ImageIO.write(image, "PNG", s.getOutputStream());

                    s.close();
                    ss.close();
                }
            }

    }

}
```

ClienteTCP

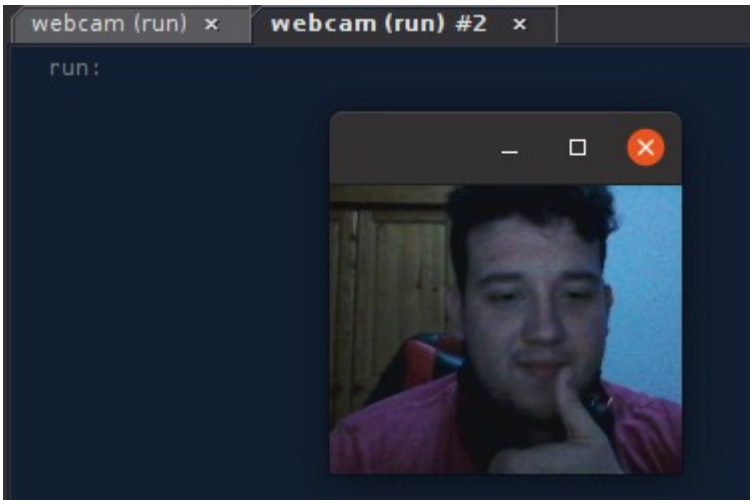
```java
import java.awt.image.BufferedImage;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;

import java.net.*;
import java.io.*;
import javax.imageio.ImageIO;


public class ClienteTCP {

    public static void main(String args[]) throws UnknownHostException, IOException {
        Socket cs = new Socket("127.0.0.1", 8890);
        //manda petición al servidor el cual devuelve la captura de la imagen de la webcam
        //que es leida por el cliente

        BufferedImage image = ImageIO.read(cs.getInputStream());
        JLabel l = new JLabel (new ImageIcon(image));
        JFrame ventana = new JFrame();
        ventana.setSize(500,500);
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ventana.getContentPane().add(l);
        ventana.pack();
        ventana.setLocation(300,300);
        ventana.setVisible(true);

        cs.close();
    }

}
```

Resultados ejecución

Ejecutamos primero el servidor que estará siempre escuchando las peticiones de los clientes.

```
run:
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Java HotSpot(TM) 64-Bit Server VM warning: You have loaded library /tmp/BridJExtractedLibraries4703200304636574819/libbridj.so which might have
It's highly recommended that you fix the library with 'execstack -c <libfile>', or link it with '-z noexecstack'.
Not enough args for null OpenIMAJGrabber.startSession(int, int, double)
Esperando...
```

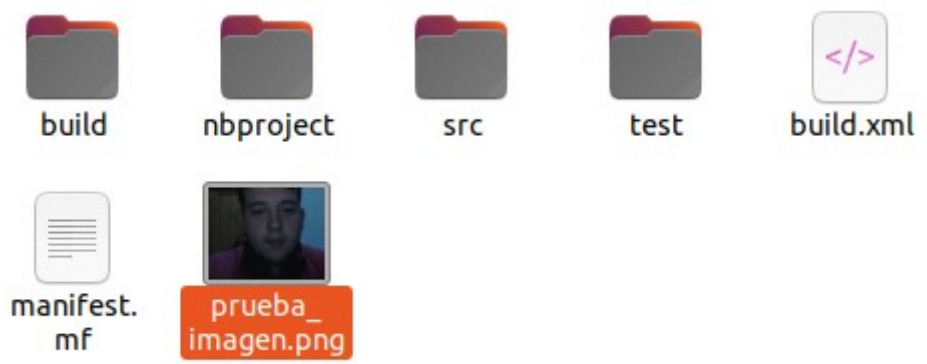
Ya se ha encendido la webcam y está esperando la petición del cliente para hacer la captura y enviarla.



Como vemos si ejecutamos el cliente, realiza la captura y nos manda la salida por ese puerto. También podemos cambiarlo e indicarle que nos guarde la imagen en un nuevo archivo y se guardará por defecto en la carpeta del proyecto.

```
BufferedImage image = webcam.getImage();
ImageIO.write(image, "PNG", new File("prueba_imagen.png"));
```

Si lo volvemos a ejecutar veremos que aparece el nuevo archivo con la imagen.



Servicio telemático mediante sockets UDP

La principal diferencia entre UDP y TCP es que TCP sí es orientado a conexión, mientras que UDP no lo es. TCP necesita saber quién está conectado a él para poder lanzar mensajes, etc. UDP, por otro lado no necesita saber esto.

ClienteUDP

```
22  public class ClienteUDP {
23
24      /**
25       * @param args the command line arguments
26       */
27      public static void main(String[] argumentos) throws UnknownHostException, IOException{
28
29          int puerto = 8899;
30          String direccionServidor = "127.0.0.1";
31          // por si queremos especificiar la dirección IP y el puerto por línea de comandos
32          if (argumentos.length == 2){
33
34              direccionServidor = argumentos[0];
35              puerto = Integer.parseInt(argumentos[1]);
36
37          }
38
39          //Creamos el cliente
40
41          new ClienteUDP(direccionServidor, puerto);
42
43      }
44
45      private ClienteUDP(String direccionServidor, int puerto) throws SocketException, UnknownHostException, IOException{
46          // Paso 1. Creación del socket
47
48          DatagramSocket s = new DatagramSocket();
49          InetAddress dir = InetAddress.getByName(direccionServidor);
50          //Paso 2. Envío de mensaje al servidor para pedir imagen
51
52          String msg = "capta imagen";
53          byte[] datos;
```

```
datos = msg.getBytes();

DatagramPacket sendP = new DatagramPacket(datos, datos.length,dir, puerto);

s.send(sendP);

//Paso 3.Recibimos el número de datagramas a recibir
//Paso 4.Recibimos la información del servidor y la almacenamos
//Paso 5.Crearemos un stream de bytes ByteArrayInputStream para poder ir leyendo la informacion recibida
//Paso 6.Leemos número de filas
//Paso 7.Leemos número de columnas

byte[] bufferdatos = new byte[2048];

DatagramPacket receiveP = new DatagramPacket(bufferdatos, bufferdatos.length);
s.receive(receiveP);
bufferdatos = receiveP.getData();

int n =0, fils =0, cols = 0;

ByteArrayOutputStream b = new ByteArrayOutputStream();

byte[] aux = new byte[4];

for (int i = 0; i < bufferdatos.length; i += 4){
    aux[0] = bufferdatos[i];
    aux[1] = bufferdatos[i+1];
    aux[2] = bufferdatos[i+2];
    aux[3] = bufferdatos[i+3];
```

```
    switch (i){
        case 0:
            n = byteArrayToInt(aux);
            break;
        case 4:
            fils = byteArrayToInt(aux);
            break;
        case 8:
            cols = byteArrayToInt(aux);
            break;
        default:
            b.write(aux);
    }
}

System.err.println("Datagramas ->" + n);
System.err.println("Nº filas->" + fils);
System.err.println("Nº columnas ->" + cols);

int i = 0;
//comprobamos y recorremos  el numero de datagramas y vamos escribiendo los datos

while (i < n){
    s.receive(receiveP);
    bufferdatos = receiveP.getData();
    b.write(bufferdatos);
    i++;
}

byte[] imgtoB = b.toByteArray();
int[] image = new int[imgtoB.length /4];
```

```
        //Paso8. Copiamos los pixeles recibidos a un objeto imagen

        BufferedImage imagen = new BufferedImage(cols,fils, BufferedImage.TYPE_INT_RGB);
        imagen.setRGB(0,0, cols, fils, image, 0, cols);

        //Paso 9 representamos la imagen
        //Creamos una ventana para mostrarla como hicimos en TCP

        JLabel l = new JLabel (new ImageIcon(imagen));
        JFrame ventana = new JFrame();
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ventana.getContentPane().add(l);
        ventana.pack();
        ventana.setLocation(300,300);
        ventana.setVisible(true);


    }

    public static int byteArrayToInt(byte[] by) {

        if (by.length == 4)
            return by[0] << 24 | (by[1] & 0xff) << 16 | (by[2] & 0xff) << 8 | (by[3] & 0xff);
        else if (by.length == 2)
            return 0x00 << 24 | 0x00 << 16 | (by[0] & 0xff) << 8 | (by[1] & 0xff);
        else if (by.length == 1)
            return by[0];
        else
            return 0;
    }
```

ServidorUDP

```
13
14
15  import java.net.*;
16  import java.io.*;
17  import java.util.Arrays;
    import javax.imageio.ImageIO;
19  /**
20   *
21   * @author manureina
22   */
23  public class ServidorUDP {
24
25
26      public static void main(String[] argumentos) throws SocketException, IOException, InterruptedException {
27
28          int puerto = 8899;
29
30          // por si queremos especificiar el puerto por línea de comandos
31          if (argumentos.length > 0){
32
33
34              puerto = Integer.parseInt(argumentos[0]);
35
36          }
37
38          //Creamos un objeto de la clase y le pasamos el puerto donde escuchará
39
            new ServidorUDP(puerto);
41
42      }
43
44      private ServidorUDP(int puerto) throws SocketException, IOException, InterruptedException{
```

```
byte[] receiveDat = new byte[2048];

Webcam webcam = Webcam.getDefault();
webcam.open();

//Paso 1. Creamos socket

DatagramSocket ss = new DatagramSocket(puerto);

//Ponemos al servidor a esperar peticiones

while (true){
    //Paso 3. Almacenamos IP y puerto en el datagrama que se enviará.

    DatagramPacket receiveP = new DatagramPacket(receiveDat,receiveDat.length);
    ss.receive(receiveP);

    String msg = new String(receiveP.getData());
    System.err.println("La solicitud ha sido recivida: " + msg);

    InetAddress direccion = receiveP.getAddress();
    int port = receiveP.getPort();


    //Paso 4. Encendemos la cámara y capturamos la imagen

    BufferedImage imagen= webcam.getImage();

    //Paso 5. Copiamos la imagen a un stream. Para ellos necesitamos
```

```
76
77          ByteArrayOutputStream bs = new ByteArrayOutputStream();
78
79          int fils = imagen.getHeight();
80          int cols = imagen .getWidth();
81          int pixeles[] = imagen.getRGB(0, 0, cols, fils, null, 0, cols);
82
83          int n = (Integer.BYTES *2 + pixeles.length * Integer.BYTES) / 2048;
84
85          //pasamos a bytes
86          bs.write(toBytes(fils));
87          bs.write(toBytes(cols));
88          bs.write(toBytes(n));
89
90          for(int i : pixeles)
91              bs.write(toBytes(i));
92
93          while (bs.size() % 2048 != 0){
94              bs.write((byte)0);
95
96          }
97
98          byte[] bufferdatos = bs.toByteArray();
99
100         //Paso 7. Transmitimos el número de datagramas que se van a enviar
101         //Paso 8. Transmitimos los datagramas que contienen los datos de la imagen
102
103         for(int i = 0; i < bufferdatos.length; i+=2048){
104             byte t[] =Arrays.copyOfRange(bufferdatos,i, i+2047);
105             DatagramPacket p = new DatagramPacket(t, t.length, direccion, port);
106             ss.send(p);
                Thread.sleep(10);
```

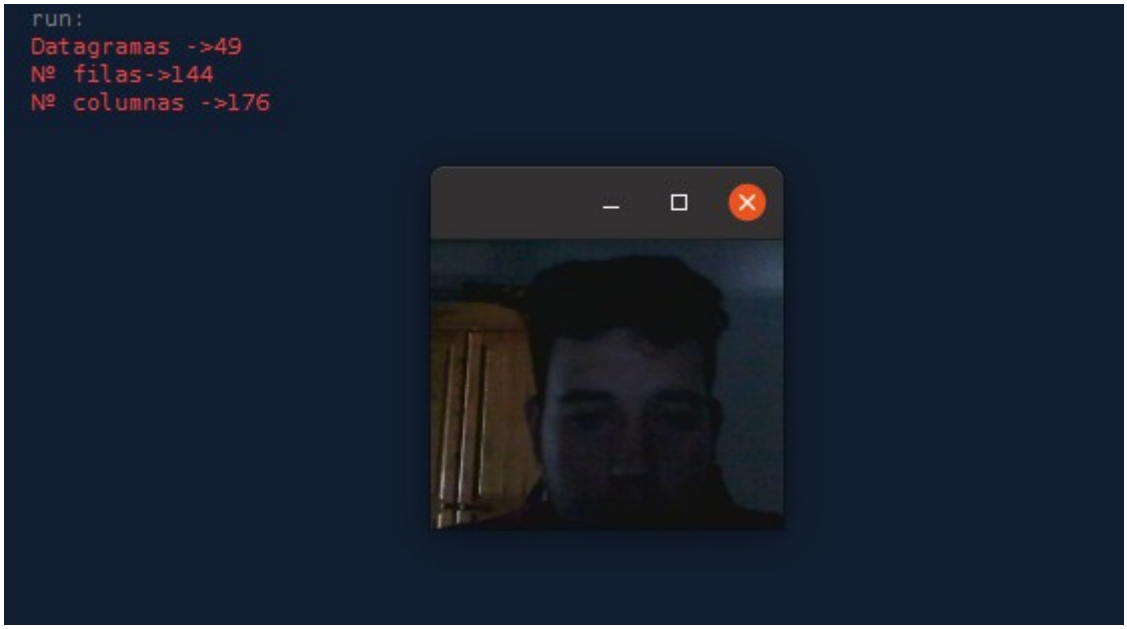
```
    public static int byteArrayToInt(byte[] by) {

        if (by.length == 4)
            return by[0] << 24 | (by[1] & 0xff) << 16 | (by[2] & 0xff) << 8 | (by[3] & 0xff);
        else if (by.length == 2)
            return 0x00 << 24 | 0x00 << 16 | (by[0] & 0xff) << 8 | (by[1] & 0xff);
        else if (by.length == 1)
            return by[0];
        else
            return 0;
    }


    public static byte[] toBytes(int i){

        byte[] resultado = new byte[4];

        resultado[0] = (byte) (i >> 24);
        resultado[1] = (byte) (i >> 16);
        resultado[2] = (byte) (i >> 8);
        resultado[3] = (byte) (i /*>>> 0*/);
        return resultado;
    }

}
```

Resultados ejecución:

Ejecutamos primero el servidor en una máquina. Encenderá la webcam y cuando reciba una petición del cliente capturará la imagen y la enviará en forma de píxeles al cliente el cual tendrá que transformarlos y convertirlos en un objeto imagen.

```
run:
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Java HotSpot(TM) 64-Bit Server VM warning: You have loaded library /tmp/BridJExtractedLibraries1965705038554487768/libbridj.so which might hav
It's highly recommended that you fix the library with 'execstack -c <libfile>', or link it with '-z noexecstack'.
Not enough args for null OpenIMAJGrabber.startSession(int, int, double)
La solicitud ha sido recivida: capta imagen
```

Cuando ejecutamos el cliente vemos que se nos muestra los datagramas necesarios para enviar la imagen, además del número de filas y columnas de la imagen. El servidor envía la imagen al cliente que recibe los píxeles de la imagen y la convierte a un objeto imagen, la representa y nos la muestra por salida.



Ejemplo de servidor concurrente (Opcional)

Con un servidor concurrente vamos a poder atender a varios clientes al mismo tiempo . Esto lo haremos creando un nuevo proceso cada vez que un cliente llegue a pedir un servicio, por lo que mientras atiende el servidor este sigue siempre escuchando.

TCPClienteConcurrente

```java
public class ClienteConcurrenteTCP {


    public static void main(String[] args) throws IOException {

        BufferedImage image;
        Socket s;

        for(int i = 0 ; i < 5 ; i++){
            s = new Socket ("127.0.0.1", 8890);

            image = ImageIO.read(s.getInputStream());
            System.out.print("Ha sido recibida: ");

            JLabel l = new JLabel (new ImageIcon(image));
            JFrame ventana = new JFrame();

            ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            ventana.getContentPane().add(l);
            ventana.pack();
            ventana.setLocation(300,300);
            ventana.setVisible(true);


            s.close();
        }

    }
```

TCPServidorConcurrente

```java
 * @author manureina
 */
public class ServidorConcurrenteTCP extends Thread {

    Socket id;

    public ServidorConcurrenteTCP (Socket s){
        id = s;
    }


    @Override
    public void run(){

        try{
            Webcam webcam = Webcam.getDefault();
            webcam.open();

            PrintWriter out = new PrintWriter(id.getOutputStream(), true);


            System.out.print("Una conexión entrante...\n");
            BufferedImage image = webcam.getImage();

            String data = "Ejecutado en thread: " + Thread.currentThread().getName();
            System.out.print("Enviando:" + image + "\n");


            ImageIO.write(image, "PNG", id.getOutputStream());


            out.print(data);
            out.close();
```

```
        }catch (IOException e){}
    }
    public static void main(String[] args) throws IOException{

        ServerSocket ss = new ServerSocket(8890);

        while(true)
            new ServidorConcurrenteTCP(ss.accept()).start();

    }
}
```

Creo que no he conseguido que funcione como debería, pero quería mostrar al menos el intento realizado.

Resultado en el que vemos como el cliente recibe las diferentes imágenes que solicitó al servidor.